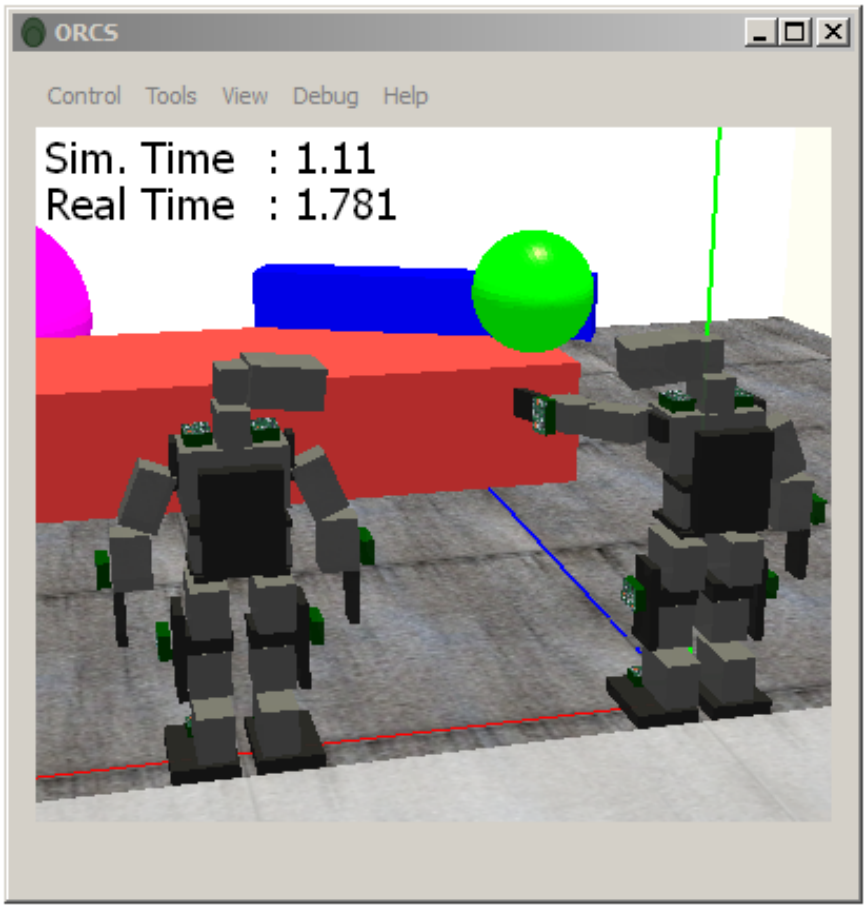
Open Robot Control and Simulation Library

– ORCS –

User Manual

Version 1.0
(September 2008)

Christian Rempis
Verena Thomas



Contents

1 Introduction **2**

2 ORCS User Manual **3**

2.1 Installation 3

2.2 Graphical User Interface 5

2.2.1 Camera Control and Keyboard Features 5

2.2.2 Control Menu 6

2.2.3 Tools Menu 8

2.2.4 View Menu 12

2.3 XML Environment Description V 1.0 13

2.4 Working with the Simulator 17

2.4.1 A-SeriesSimulator Basic Usage 17

2.4.2 Interaction with the MotionEditor 18

2.4.3 A-SeriesClientSimulator Synchronous Control 20

1 Introduction

The *Open Robot Control and Simulation Library* (ORCS) is developed as part of the ALEAR project(http://www.alear.eu). The current library offers a physical simulator that can be used to simulate humanoid robots, but in the future the library will be heavily extended to support:

- the definition of different robot architectures;
- an application to generate neural networks to control the simulated agents;
- an evolution framework to evolve neural networks using different evolutionary algorithms.

ORCS aims to become a versatile robot simulation environment that is easy to use for different types of robots. Including this goal, an easily usable mechanism of defining own motors and sensors is planned. Version 1.0 of ORCS is based on the *Open Dynamics Engine* (ODE) – a known, open source physics engine – but due to a physical abstraction layer, the physics library can be exchanged with little effort. This offers the possibility of trying new physics engines. In addition to ODE, the promising *Impulse-based dynamic simulation* engine by J. Bender [2] is already included into ORCS in large part.

This manual presents the existing tools and features of the applications created with the ORCS library and explains their installation and use. More information and the latest version of the software are available on the ORCS home page (http://www.cogsci.uni-osnabrueck.de/ neurokybernetik/tools/orcs).

2 ORCS User Manual

2.1 Installation

Windows

Windows is supported with pre-build binaries. To install the simulator the A-SeriesSimulator.zip file has to be downloaded and unpacked.

The zip file contains the following files and directories:

- the simulator executable for interactive usage. (`A-SeriesSimulator`)
- the simulator executable for usage with the evolution system. (`A-SeriesClientSimulator`)
- the directory `orcsconf` which contains configuration files and resources of the simulator.
- an application to demonstrate the usage of the communication protocol. (`ORSC_OSP_SliderControl`)
- the `MotionEditor` to create and execute motion networks.
- a folder containing files for the described demonstration scenarios, like motion network files, parameter settings and environment files.
- a number of batch files that start the simulator with specific scenarios, described in this manual.

The executables require the Qt shared libraries to be on the library path. If not installed, they have to be obtained from the homepage of Qt [7]. Alternatively the libraries can be downloaded from the ORCS home page [1] and be placed in the executable folder manually.

Linux (Binary)

For some Linux distributions (Ubuntu 8.04 up) binary packages can be downloaded on the web page. These zip files contain the same content as the window version except of the `MotionEditor` (section 2.4.2), which is only available for Windows.

The executables require the Qt libraries to be on the library path. On Ubuntu and most Debian based systems, these libraries can be installed as superuser via apt-get:

```
sudo apt-get install libqt4-core libqt4-dev libqt4-gui libqthreads-12
```

On the supported Linux distributions the simulator should run out of the box.

Linux (Build from Source)

For all other Linux systems the simulator has to be built from source code.

The source code can be obtained at the ORCS home page. In addition to the simulator source code the physics libraries ODE and/or IBDS are required (currently only ODE in version 0.9 is fully supported). The physics library has to be compiled first. Then the header files and the static physics library have to be put on the build path. Also common build tools, like qmake (contained in the `libqt4-dev` package), make, java and g++ are required.

The required packages can be installed using apt-get:

```
sudo apt-get install sun-java6-jre g++ automake autoconf libqt4-core
                                libqt4-dev libqt4-gui libqthreads-12
```

The ODE physics library (Version 0.9) can be obtained from the ODE homepage (http://www.ode.org). It should be unpacked and compiled with double-precision support:

```
cd ODE_BUILD_DIR
sh autogen.sh
./configure --enable-double-precision --enable-release
make
sudo make install
```

Then a new directory has to be created at the same level as the ORCS source directory named `ode-0.9`, with a subdirectory named `lib`. The compiled static ODE library `libode.a` has to be copied into the `lib` folder:

```
cd BUILD_DIR
mkdir ode-0.9
cd ode-0.9
mkdir lib
cp ODE_BUILD_DIR/ode-0.9/ode/src/libode.a ./lib
```

When the prerequisites are fulfilled, the ORCS prepare tool `qmakeprepare.jar` should be started in the ORCS source directory. The prepare tool modifies the project files to build the ORCS sources with ODE support. In a later release other physics libraries might be enabled with the prepare tool before compilation. Enter qmake and make to build all programs of the library.

```
cd BUILD_DIR/orcs
java -jar qmakeprepare.jar
qmake
make
```

This builds the library including the application `KeyFrameSniffer` and the simulators `A-SeriesSimulator` and `A-SeriesClientSimulator`.

2.2 Graphical User Interface

The ORCS simulator can be started with a graphical user interface (GUI). It provides an OpenGL visualization of the simulation environment and a set of additional tools and features to manipulate different aspects of the simulation. This section will introduce the most important features and their usage. More information on using ORCS and its GUI can be found in the ORCS help (`F1` or `Help` menu).

All values representing entities are following the SI - International System of Units. Furthermore angles are specified in degrees. This simplifies working with the simulation settings. Internally radians and quaternions are used for the physical simulation.

The main window (figure 1) offers a view on the simulation environment where the keyboard and mouse can be used to change the viewing position. In addition further tools are available which can be opened using the menu or the corresponding keyboard command.

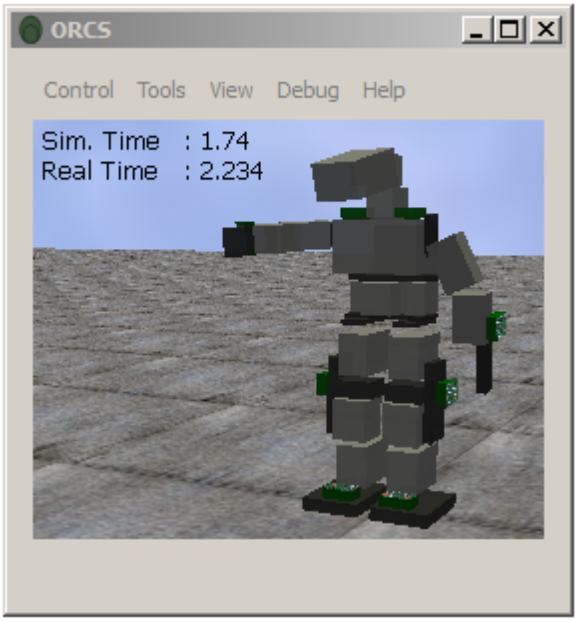


Figure 1: ORCS main window showing an A-Series robot. The menu above the visualization offers tools to manipulate and control the simulation.

2.2.1 Camera Control and Keyboard Features

ORCS offers a hybrid control of the viewpoint that uses keyboard commands as well as mouse manipulations. Table 1 shows an overview of all mouse and keyboard commands and their impact on the visualization. As the user can define multiple cameras that show the same scene from different positions (see section 2.2.4), each camera can have an individual camera control that is active as soon as the respective camera visualization window has the focus.

Keyboard / Mouse	Action
right mouse button	Zooming in/out and translational left/right movement.
left mouse button	Rotational left/right movement and rotational up/down movement. Changes of the orientation result in a rotation around the current viewpoint.
middle mouse button	Translational up/down movement.
`a` or `left arrow`	Translational left movement.
`d` or `right arrow`	Translational right movement.
`w` or `arrow up`	Zooming in.
`s` or `arrow up`	Zooming out.
`e` or `page up`	Translational up movement.
`q` or `page down`	Translational down movement.
`Shift`	Accelerates all viewpoint movements.
`Ctrl + Shift + v`	Save the current viewpoint. This will override the previously stored viewpoint.
`Ctrl + v`	Restore the saved viewpoint.
`Ctrl + x`	Show/hide textures.
`Ctrl + t`	Show/hide time information, which shows the passed simulation time and real-time since the last reset.
`Ctrl + a`	Show/hide coordinate axes.
`Ctrl + g`	Show/hide a grid with a edge length of 1m.

Table 1: List of all mouse and keyboard commands to control the viewpoint of a camera or the visualization.

2.2.2 Control Menu

The `Control` menu offers several possibilities to manipulate the simulation. ORCS simulators using the Orcs Standard Protocol (OSP) [4] do not provide the control menu, as the simulation control is synchronously triggered by ORCS OSP clients.

Motor Control Manager. To enable an easy manipulation of the simulated agents, slider controls for all motors of each agent are available on demand. The `Motor Control Manager` shows a list of all known agents, which can be manipulated by sliders (figure 2). For each chosen agent, a separate window will be opened, which offers a slider for each motor of the agent.

The name of the agent is added to the list below the two buttons. Double clicking on a name will hide/show the according motor control window. A control window can be deleted with the `Remove Controls` button.

To allow for repeating movements and to manipulate several motors simultaneously, a save and restore mechanism is included into each slider panel. The keys $[0-9]$ can be used to switch between previously saved motor settings.

Using `Ctrl+`$[0-9]$ the currently chosen motor values are stored internally and related with the according number key. By pressing the according key, the previously saved combination of desired motor positions is restored.

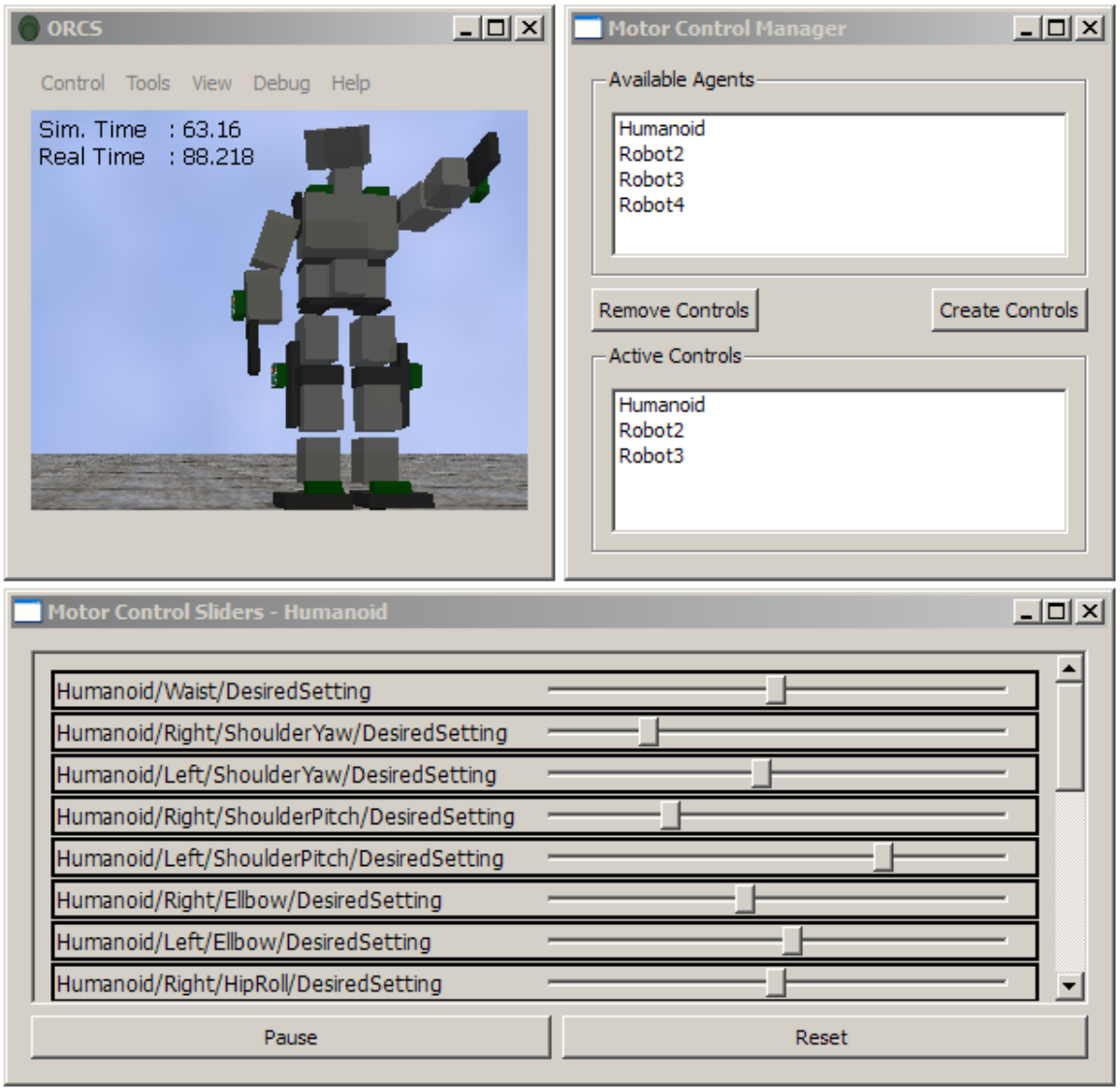


Figure 2: The `Motor Control Manager` (upper right) holds a list of available agents and allows the creation of control panels, which provide a list of sliders to manipulate the motors of the agent. At the bottom an example of a slider control is shown.

Realtime. Runs the simulation in real-time. If the used machine is capable of running the simulator faster than read-time, this will reduce the speed and adapt it to real-time. If the machine is too slow to reach real-time, enabling `Real-time` will result in running the simulation at maximum speed.

Pause. Pauses or resumes the simulation.

Next Step. If the simulation is paused, this action triggers a single simulation step. This allows the user to step through the simulation.

Reset Simulation. Resets the simulation to the initial conditions. During startup ORCS saves all physically necessary information of each object and joint. By calling reset, the physics will be deleted and newly created using the previously saved information. By using the `Object Properties` tool (section 2.2.3), the initial information for a particular value can be changed.

2.2.3 Tools Menu

The simulator provides a large set of parameters that can be modified or observed by the user. There are two types: simulation properties and object properties.

Simulation Properties. The `Simulation Properties` tool (see figure 3) allows direct access to all available parameters of the variable repository, such as the settings of the physics engine, the simulation delay and step size, or parameters of the graphical user interface. The global settings for the motors and sensors and all object parameters are available in this tool as well. The latter parameters are additionally accessible by the `Object Properties` tool, where plotting and logging mechanisms are available.

By using regular expressions in the upper text field the user can search for parameters. All matching parameters will be shown in the drop down box below and can be selected with the mouse. To create visualizations for the current selection of parameters, the `Add` button or the `return` key can be used. As long as the checkbox of a parameter is checked, the value of that parameter is automatically updated when it was changed somewhere else in the simulator.

A modification of the parameter value is possible by typing a new value into the according text field and pressing the `return` key. To apply several changes simultaneously the `Apply` button can be used instead of repeatedly pressing the return key.

A selected set of parameters can be saved to a file. Such parameter files can be loaded at startup with `-ival` or `-val` (section 2.4.1), or can be loaded at runtime with the Simulation Properties Tool. In the latter case loaded values are not applied directly to prevent undesired changes. Instead they have to be applied manually with the `Apply` button.

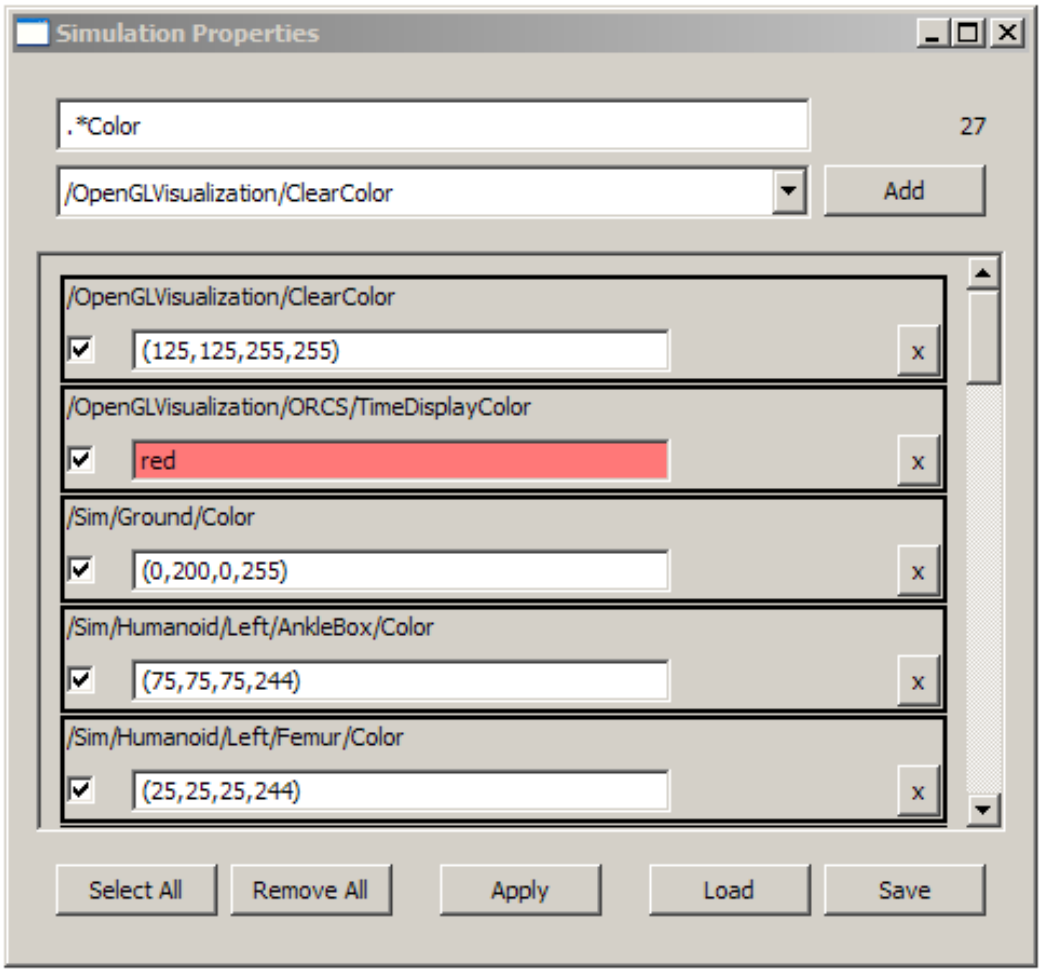


Figure 3: The `Simulation Properties` tool. All parameters matching the current regular expression in the upper text field will be shown in the lower part of the panel. A text field offers the possibility of observing and modifying the value of the individual parameters. If the user changed the value of a parameter, the according text field will be highlighted (see figure) until the modification was applied. The buttons below provide additional features like saving and loading of value collections.

Object Properties. The `Object Properties` tool enables an observation and manipulation of object properties (figure 4). These properties describe states, settings or attributes of all simulated objects, like motors, sensors, body parts or environment objects. The selection mechanism is similar to the `Simulation Properties` tool, so the user can search for desired properties using regular expressions. In contrast to the `Simulation Properties` tool, several tabs can be created (using the `+/-` buttons at the top), where independent sets of object parameters can be managed. The possibility of creating separate collections is important, because apart from observing and manipulating the property values, these collections can also be logged to files or plotted in graphs.

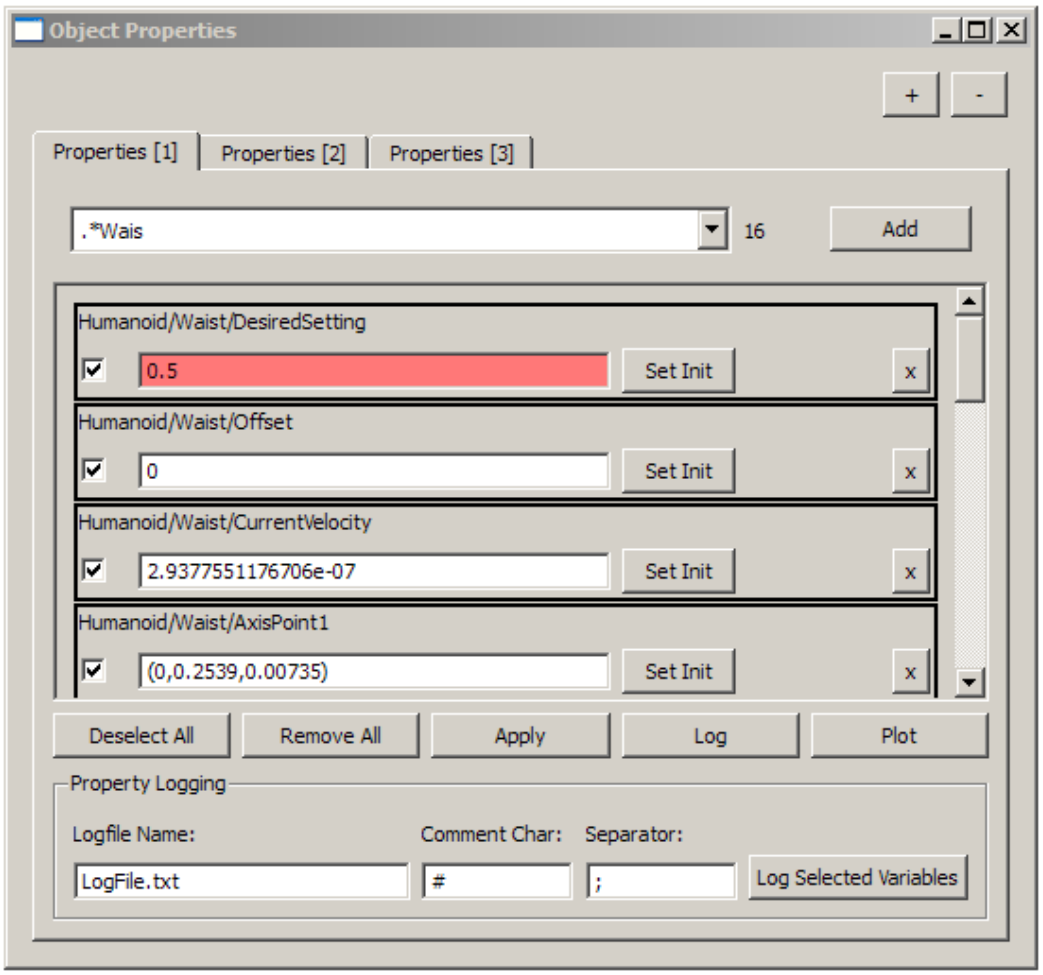


Figure 4: The `Object Properties` tool. Several tabs can be created to compose different parameter sets that can be observed and manipulated. As in the `Simulation Properties` tool, values modified by the user are highlighted until they were applied. The buttons at the bottom provide additional features like a `ValuePlotter` and a `Logger`. Below the buttons, the `Logger` panel is visible. On starting the `Object Properties` tool, the `Logger` is hidden and opens only after pressing the `Log` button.

Parameters in the `Object Properties` tool can be changed manually as in the `Simulation Properties` tool. Most parameters however will be overwritten at a reset with the default

settings found at startup of the simulation. To keep a setting even at a simulation reset, the `SetInit` button has to be pressed to transfer the value to the initial settings.

Value Plotter. By pushing the `Plot` button a plotter opens and starts to record and plot the history of all currently selected parameters (figure 5). This is useful for observations of the motors and sensors of the robot during complex motions.

If the selection of values in the `Object Properties` panel changes while a plotter window is open, the `Plot` button needs to be pressed again to update the selection of plotted parameters. As the plotter visualizes the development of parameters over time only numerical values (double, int, vector3d) can be plotted.

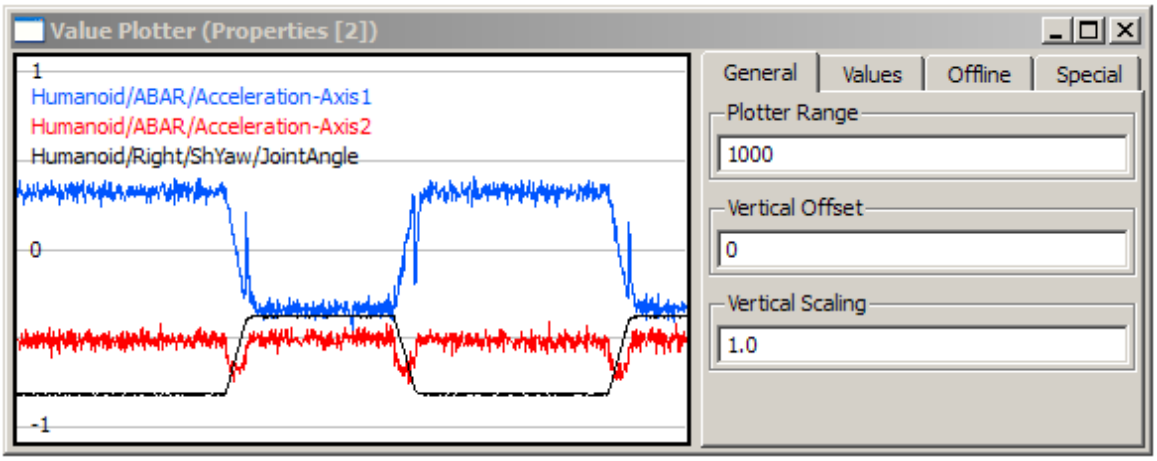


Figure 5: The `Value Plotter` shows the time development of the currently selected parameters of the `Object Properties` tool. The tabs on the right side offer feature as: color selection, hide/show single parameters and loading offline data.

The Plotter Window has several tabs to manipulate the Value Plotter:

- **General.** The tools in the `General` tab allow to vertically scale and move the plotted values and to change the history size of the plotter.
- **Values.** The `Values` tab shows a list of all currently plotted parameters and enables to change the name, color and visibility of each parameter.
- **Offline.** To compare the observed parameters to a previously recorded set of parameters (e.g. recorded with the logger), a logged property sequence can be loaded in the `Offline` tab of the Plotter. The loaded sequences can be scaled and moved independently of the values plotted on-line to find a match between the data for direct comparisons.
- **Special.** If the number of plotted values becomes large and the legend becomes disturbing, the checkbox in the `Special` tab can be used to hide the legend.

Logger. During simulation the progression of the currently selected properties can be logged to a file for later analysis (see figure 4).

Using the three text fields that become visible on pressing the `Log` button, the file name for the log, a comment character and a separator character can be set to simplify the usage of the data in external applications, like OpenOffice or Matlab. The logger adds the names of the logged properties as comments to the start of the file, using the comment character as prefix. Logging starts by pressing the `Log Selected Variables` button. Hereby the values of all selected properties are written to the selected file after each simulation step, using the separator character to separate the values. The logger can be stopped with the `Log Selected Variables` button.

2.2.4 View Menu

Camera Manager. The `Camera Manager` (figure 6) is a tool for creating and managing different views on the simulation. The manager allows to choose a reference object in the simulation, to which the new camera view will be attached. The initially chosen position and orientation values are offset values from the reference objects position and orientation. If the reference object is moved, the camera view is moved accordingly and the view on the scene changes.

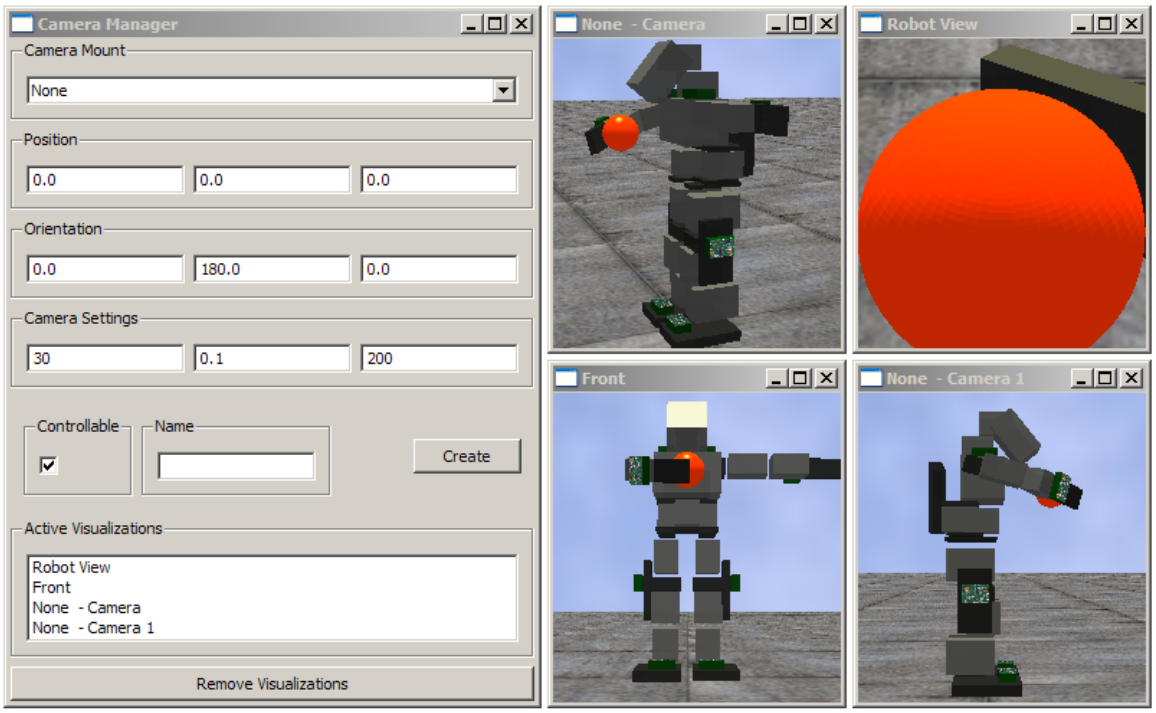


Figure 6: The `Camera Manager` consists of a property selection area and a list of existing, active views. Some examples of different views of the same scene are shown at the right.

The `Controllable` checkbox indicates, whether a camera control is available for a newly created view. If keyboard and mouse control is enabled, then movements are always relative to the reference objects position and orientation.

2.3 XML Environment Description V 1.0

For experiments requiring the agent to interact with its environment or other robots, an easy way to define such an environment is needed. Especially for the language games multiple objects of different shape, size and color have to be defined and modified easily.

To avoid the necessity of any deeper knowledge about the simulator and its architecture, a XML-description for simple environment objects is provided. A set of environment objects can be loaded at startup from an XML file. The XML file can be specified with the command line option `-env ⟨environmentfile⟩.xml`. All objects defined in the according file will be added to the simulation.

Besides the possibility of loading environment objects into the simulation, it is as well possible to add further A-Series robots.

The XML example below (`exampleEnvironment.xml`) shows a full environment description that adds two environment objects and places an additional agent into the simulation. The result of loading this XML-description is shown in figure 7.

```
<ORCS version="1.0">
 <Environment>
  <object type="box"
          name="FirstBox"
          position="(0.025,0.075,0.25)"
          orientation="(45,45,45)"
          dimension="(0.1,0.1,0.2)"
          color="(0,100,100,200)"
          texture="AccelBoard"/>
  <object type="sphere"
          name="SecondSphere"
          position="(-0.255,0.5,0.2)"
          color="red"
          dynamic="true"
          mass="0.05"
          material="ABS"
          dimension="0.05"/>
 </Environment>

 <Agent>
  <ASeries name="A-Series" position="(0.3,0.02,-0.3)" />
 </Agent>
</ORCS>
```

The <Environment> node of the XML definition allows for the definition of sphere- and box-objects of different size and color. Each <object> tag marks one object definition, where several attributes are required to describe the properties of the environment objects. A list of all parameters, their meaning and their usage can be found in table 2 (object parameters) and in table 3 (agent parameters).

Parameter	Mandatory	Usage
type		The `type` defines the shape of the environment object and can have the value `box` or `sphere`.
name	Optional	All objects and joints in the simulation carry names. The name can be used during simulation to manipulate the properties of an object using the `Object properties` tool (section 2.2.3), or to change parameters before starting the simulation by loading a value-file (section 2.4.1). If no `name` parameter is defined, default names (`box` and `sphere`) will be used.
position		The `position` defines the location of the object within the simulation environment in global coordinates.
orientation	Optional	The `orientation` parameter sets the orientation of the environment object in degrees around the three coordinate axes (Euler angles).
dimension		The `dimension` parameter defines the type specific shape and size properties. For sphere objects the dimension is a double value, representing the radius of the sphere. For box objects the dimension defines the width, height and depth.
material	Optional	To manipulate the collision properties of the environment object material information can be added (see section 2.3 for more information). If no `material` is chosen, the material `Default` is used.
color	Optional	The `color` parameter allows to modify the color of the environment object. There are some predefined color values that can be chosen by naming the color (e.g. blue, darkYellow). If non-standard colors need to be used an RGBA-representation of the desired color can be used. If no color is specified, white will be used as default.
texture	Optional	The `texture` defines the name of the texture image to be used for the visualization (see section 2.3 for more information). If no texture is chosen, `Default` will be used.
dynamic	Optional	As default all environment objects that are added to the simulation are static. Gravitation will not act on them and during collisions, their position will not be changed. To add an environment object, which can be manipulated, the parameter `dynamic` needs to be set to true. All dynamic environment objects need to additionally define a `mass`.
mass	if dynamic	For static objects no mass information is required. For dynamic objects the mass describes their weight.

Table 2: List of attributes and their valid values for writing XML descriptions of environment objects. Some attributes are mandatory, denoted in the second column. Knowing the name of the newly defined object, these parameters can also be modified during execution time by using the `Object Properties` tool (section 2.2.3).

Parameter	Mandatory	Usage
name	Optional	Each new agent can have a name. All parameters describing the properties of this particular agent will use this name as prefix.
position		The global start position of the agent. If set to (0,0,0) the agent will start standing at the coordinate center of the simulation.

Table 3: List of attributes and their valid values for writing XML-descriptions of A-Series humanoids.

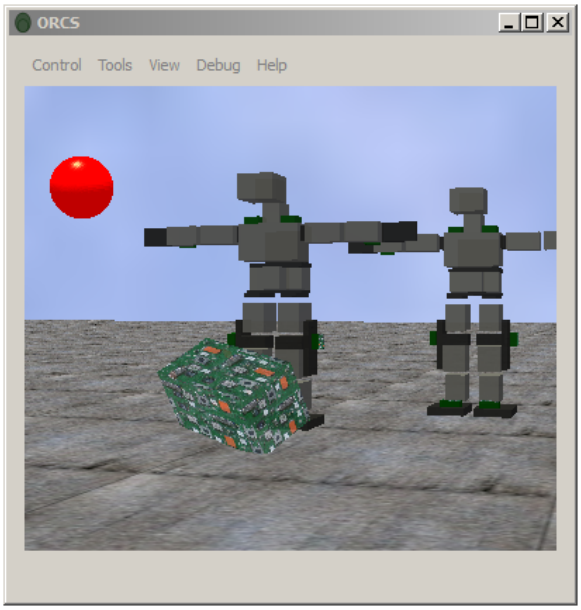


Figure 7: Simulation resulting from loading the XML example `exampleEnvironment.xml` presented in the tutorial. The file defines a box object, a dynamic sphere object and an additional A-Series humanoid.

Texture and Material Descriptions

Each simulated physical object in ORCS can be assigned with a material. This material information influences the way collisions are resolved for this object. During collision handling, the materials of the two colliding objects are determined and a lookup table is consulted to find the according friction coefficient. To change the collision behavior, the user simply needs to change the material of the object.

Currently the following materials are supported: `ABS` (Acrylonitrile butadiene styrene – plastics material used for rapid prototyping), `PVC`, `Carpet`, `Wood`, `Latex` and two different plastics from the Bioloid construction kit [5] named `PlasticsWhite` and `PlasticsGray`. There is a `Default` material that behaves as `Latex` on `Carpet` for every partner material. If the material is not set by the user the object will use this default material.

If an invalid material name or `None` is used to define the material, a second friction and elasticity calculation mechanism is used. Each object has two additional parameters to set the friction value (`DynamicFriction`) and the elasticity value (`Elasticity`). During collision handling, at first the material information is checked. If the material information of one of the objects is invalid, the above named parameters of both objects will be used to determine the friction coefficient and the elasticity of the contact. For this purpose, the friction values are summed up, whereas the elasticity constants are multiplied with each other to generate the contact properties.

The OpenGL-based visualization can use textures for every object. To influence the texture that will be applied to the object, two mechanisms are applied. Each object possesses a parameter called `texture`, where the name of an image file (without file extension) can be named. If the program could find this image during startup, the texture will be applied to the object. If no image with the according name could be found, the object will be

drawn without texture. If the texture parameter was not changed by the user, it is set to `Default`. Setting the `texture` parameter to default will result in using the name of the objects' material to determine the texture.

There are a few textures (Sky, PVC and AccelBoard) that are included into the program. To use own images for the materials or to add new textures to the program, these files need to be placed into the folder `orcsconf/textures` relative to the place, where the program is started. The supported image formats are `JPEG`, `PNG`, `TIFF`, `GIF`, `BMP` and `PPM`. Even if the simulator already provides textures with the same name, the new images – if correctly placed – will be used instead.

2.4 Working with the Simulator

2.4.1 A-SeriesSimulator Basic Usage

To use the simulator directly in an interactive mode, users can use the `A-SeriesSimulator` application by starting the executable. The simulator can be adapted by using command line arguments:

-env ⟨**environmentfile**⟩ loads an environment file (section 2.3).

-port ⟨**port**⟩ sets the port of the ODCP UDP server [3] for the direct control of the A-Series robot.

-gui / -nogui starts the application with graphical user interface enabled (default) or disabled (section 2.2).

-ival ⟨**valuefile**⟩ loads a parameter file before the initialization of the simulator takes place. This can be used to influence the initialization process. Users can use this command line argument multiple times to load more than one value file. The command line argument order is the order, in which the files are loaded.

-val ⟨**valuefile**⟩ loads a parameter file after the initialization of the simulation. This can be used to change the settings of the completely initialized simulation, e.g. to set the camera position, set motor model parameters or to change aspects of the graphical user interface.

-play ⟨**agent**⟩ ⟨**keyframerecord**⟩ installs the `KeyFramePlayer` (section 2.4.2) for the A-Series robot in the simulator and controls the robot named ⟨agent⟩ with previously recorded keyframe motions loaded from the specified ⟨keyframerecord⟩ file. Such a file can be created with the `KeyFrameSniffer` tool together with the `MotionEditor` or manually with a text editor.

Once the application is started the simulation automatically starts running. The time display in the upper left corner shows the simulation time and the real time. This indicates whether the computer is fast enough to run the simulator in real time. By default the simulator executes simulation steps with a user defined delay in between. A large delay leads to a simulation in slow motion, while a very low delay allows the simulation to run with the maximum possible performance the computer is capable of.

The delay can be adjusted by manipulating the repository variable `/Simulation/Delay` in the `Simulation Properties` tool (section 2.2.3).

The simulation can be paused and reset in the control menu. When paused, the control menu also allows for the execution of single simulation steps.

Manual Robot Control

The A-Series robot can be controlled manually with the `Motor Control Manager` (section 2.2.2) found in the control menu. This tool provides slider panels to adjust the desired angles of all motors of the A-Series robots.

Simulation and Object Properties

The simulation can be interactively modified using the property tools (section 2.2.3 and 2.2.3) found in the `Tools` menu. The `Simulation Properties` tool enables loading and saving of parameter sets, which is useful if motor or physics parameters have been experimentally determined. Such parameter sets can be saved to a file to reload the settings later. Modifications, observations and logging of simulated object properties can be done in the `Object Properties` tool. Desired settings of parameters can be transferred to the initial settings of an object by pressing the `Set Init` button of a property. Such settings will be restored whenever the simulation is reset.

Camera Views

For language games or for presentation purposes additional views on the simulated scenario can be added. The `Camera Manager` (section 2.2.4) in the `View` menu controls the creation of such views. Independent cameras can be added at global scope or be attached to simulated objects. This way each robot can for instance provide an own view from the perspective of its head camera or the user can create a camera that automatically follows a robot in a certain distance.

2.4.2 Interaction with the MotionEditor

MotionEditor. Using the `MotionEditor` [6] (figure 8) it is possible to design keyframe-based motions for the A-Series humanoid robots. A keyframe defines the desired angles of all motors. The composition of a sequence of these keyframes – connected by transitions – forms a motion net. To achieve a smooth motion, the `MotionEditor` interpolates between two successive keyframes by slowly changing the settings from one keyframe to the next one within a given duration.

One is able to connect the `MotionEditor` directly via USB to the physical A-Series robot to execute motion nets or to deploy them to the flash memory of the robot. Furthermore a connection to a simulated robot in the ORCS simulator `A-SeriesSimulation` can be established. Even a simultaneous connection to a physical and a simulated robot is possible. Connections can be established via the `Communication` menu.

The `MotionEditor` works purely graphical. Keyframes are shown as small rectangles with their keyframe number. Transitions are shown as arrows connecting keyframes. Keyframes are generated using the right mouse button on the empty drawing area and by choosing `New keyframe` in the context menu. To create a transition, action `New transition` in the context menu of a keyframe can be used. A click on another keyframe creates the transition towards the latter one. Using the context menu it is further possible to delete elements or copy and paste them. The execution of a motion net will always start with keyframe 1. For testing purposes some sample motion nets (*.mef) have been included in the ORCS package (the program and the example motion files are available in the folder `/tools/MotionEditor` of the enclosed zip-file).

The main window is split into three areas. The largest is the editor area at the left. At the right one can find the motor settings area on top and the player area at the bottom. In the editor area one can edit the keyframe net. Selecting a keyframe or a transition shows

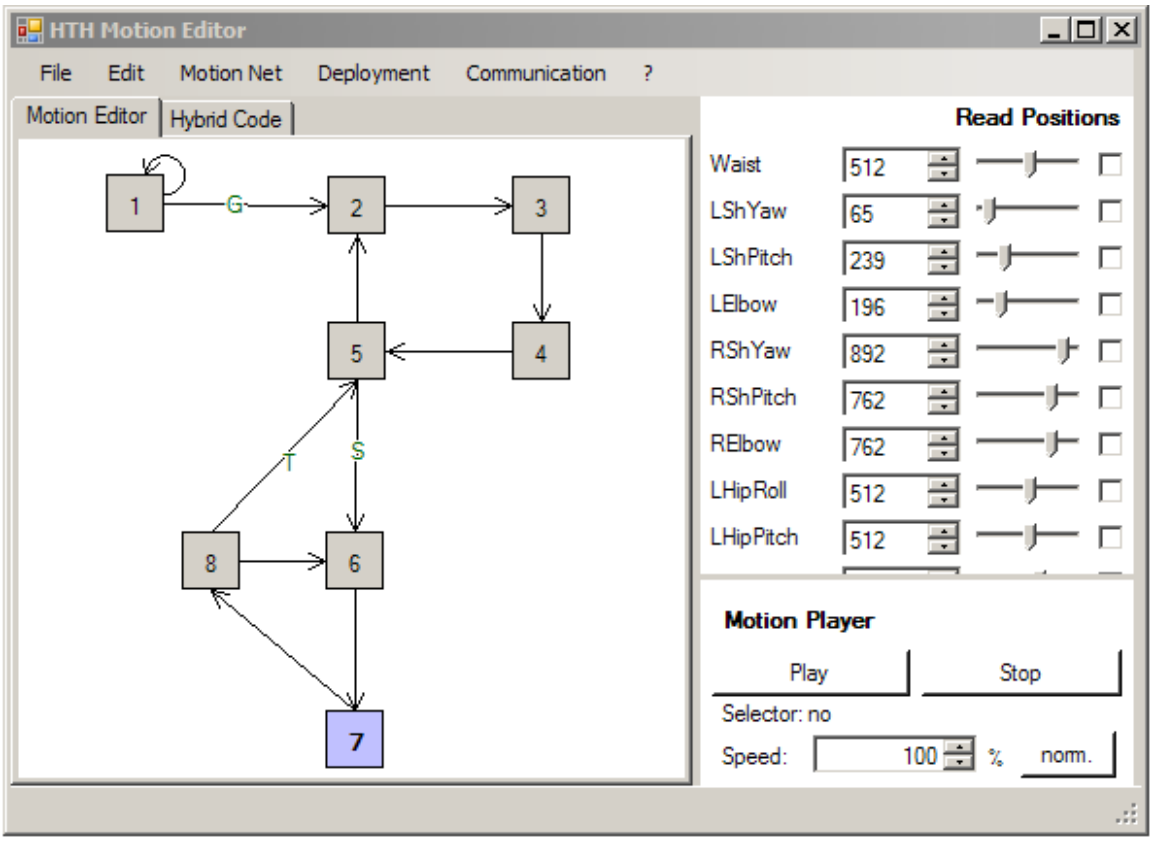


Figure 8: The MotionEditor. The graphical user interface is divided into three section: a keyframe editor area, a player area and an area to set the motor settings of keyframes.

the corresponding settings in the settings area (motor positions for keyframes and torque values for transitions). The player area is used to run the current motion net (at the robot, the simulator or both, regarding which devices are connected).

Selecting one or more keyframes (the `Shift` key enables multiple selections), motor values are shown at the right in the settings area. The sliders can be used to manipulate these settings. If a (physical or simulated) robot is connected, the current keyframe is immediately transmitted to the robot.

Selecting a transition shows the corresponding settings in the settings area. One is able to change the transition time and the torque values[1]. Motion nets can have multiple transitions starting in one keyframe. The decision, which transition is chosen while playing is made using a global `selector`, which is an ASCII character. If the global selector is equal to the selector of a transition, this transition is chosen. The selector for a transition is set via the context menu of the transition.

A loaded or created motion net can be executed (played) using the integrated player. Clicking the `Start` button starts playing from keyframe 1. The player can be controlled by changing the global selector. This is accomplished by simply pressing the corresponding key (e.g. pressing key `R` results in the global selector `R`).

[1]Currently ORCS v1.0 does not fully support torque modulations

KeyFrameSniffer and KeyFramePlayer. Instead of connecting to the ORCS simulation, the `MotionEditor` can be connected to the application `KeyFrameSniffer`. This tool provides the same interface as the `A-SeriesSimulation` application, but does not contain a simulator. The purpose of this program is to log received motion commands sent by the `MotionEditor` while running a motion network. The keyframe settings are recorded as non-interactive keyframe sequences to a file to be replayed in the simulator without a connection to the `MotionEditor`. The command-line based tool can be started with:

```
KeyFrameSniffer <fileName>
```

The filename defines the name of the file in which the logged data will be saved. The `KeyFrameSniffer` runs without graphical user interface in the background and stops automatically when the `MotionEditor` is disconnected with action `Disconnect Simulator` in the `Communication` menu.

Recorded keyframe sequences can be used to execute the exact movement sequences in the simulator without a connection to the `MotionEditor`. For this the `A-SeriesSimulation` provides a build-in `KeyFramePlayer`, which can handle the motion files. The simulator can be started with a `KeyFramePlayer` as follows:

```
A-SeriesClientSimulator -play <agent> <keyframerecord>
```

The player requires the name of the agent to be controlled (e.g. *Humanoid*) and the file <keyframerecord> with the motions to apply to the agent. Multiple agents can be controlled by individual keyframe sequences by using the `-play` parameter several times with different agent names.

The purpose and reason for the `KeyFrameSniffer` and the `KeyFramePlayer` is the requirement of the `MotionEditor` to have a real-time connection, as the transition durations are based on real-time. Due to the fact that achieving constant and reliable real-time in a simulation program is difficult, this requirement might pose problems when trying to compare very accurately the movements of the physical robot and the simulation. The `KeyFramePlayer` therefore records a timestamp for each motion command received from the `MotionEditor`. This time information is used during playback to synchronize the motion commands with the simulation time. This reliefs the simulation from the need to run in real-time, as the motion commands now are applied based on the simulation time, not the real-time. This is essential for comparable motions, especially on slower computers, where real-time can not be achieved.

2.4.3 A-SeriesClientSimulator Synchronous Control

The simulator version `A-SeriesClientSimulator` was designed to run on a computer cluster for the evolution of neuro-controllers. Therefore by default this simulator starts without graphical user interface (GUI). To enable the GUI the application has to be started with command line parameter `-gui`.

In difference to the `A-SeriesSimulator` this simulator does not immediately start with the simulation in its own simulation loop. Instead it waits for the connection of clients using

the OSP 1.1 protocol [4]. The simulation is started only if at least one client connects and triggers the simulation execution steps via UDP commands.

To start the simulator with a simple example scenario (shown in figure 9), the following line can be used:

```
A-SeriesClientSimulator -env languageGame.xml -val languageGame.val -gui
```

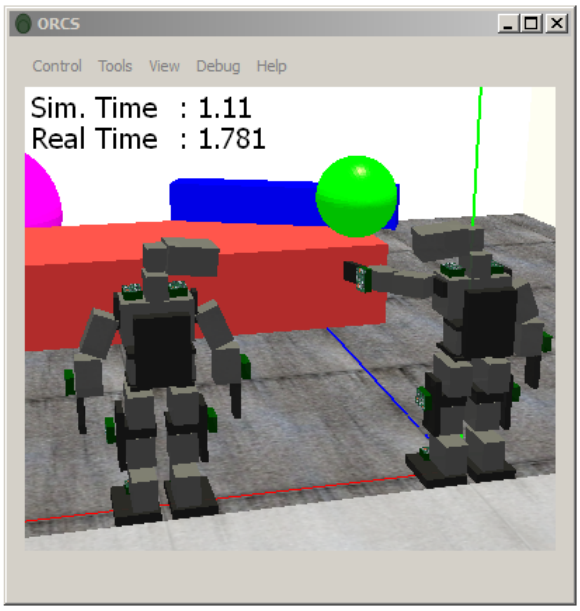


Figure 9: Result of opening the simulation with the XML-file `languageGame.xml` and the value-file `languageGame.val`.

Currently there are two applications that can be used to connect to the simulator and to control the simulation: the `ORCS_OSP_SliderControl` and the `ORCS_NetworkEvaluator`.

Application `ORCS_OSP_SliderControl`.

The `ORCS_OSP_SliderControl` application demonstrates the usage of the OSP 1.1 protocol. The source code is provided with the deliverable and can be used to build custom applications to interact with the agents in the simulator.

To use the `ORCS_OSP_SliderControl` Java is required. The application can be started with the following command:

```
java -jar ORSC_OSP_SliderControl
```

In the application window (figure 10) the `ORCS_OSP_SliderControl` can be connected to the simulator. In the area below all detected robots are shown with detailed information about their names and their motor/sensor interface. To control a robot it has to be selected in the list and the `Control Agent` button has to be pressed. By pressing it again the control of the agent is returned to the simulator. If an attempt is made to control an agent that

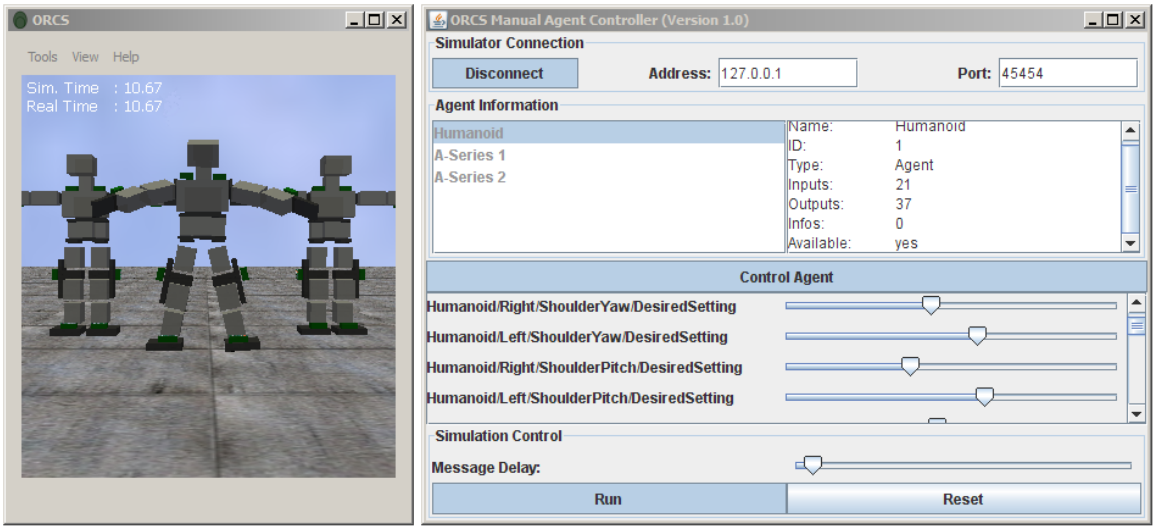


Figure 10: The application window of the `ORCS_OSP_SliderControl`. The program allows to select an agent to be controlled by sliders. Additionally informations about all existing agents are provided.

is already controlled by another client, a notification is shown and the agent control is not committed to the `ORCS_OSP_SliderControl`.

Once control is successfully taken by the `ORCS_OSP_SliderControl`, each motor of the controlled agent can be adjusted with sliders. However, since the simulation execution is client triggered, nothing will happen before all clients, that control at least one agent, request a simulation step. To do so the `Run` button has to be pressed. If more than one client is connected to the simulator, then all these clients have to start sending their step requests.

To reset the simulator all agent controlling clients have to send a reset request.

This synchronization enables the control of multiple robots in a step triggered way without the need to run all clients with exactly the same speed.

Application *ORCS_NetworkEvaluator*

The `ORCS_NetworkEvaluator` executes neural networks as controllers for a simulated agent in an `A-SeriesClientSimulator` and evaluates the network performance with a fitness function. Such an application is used on the cluster computer to evaluate neural network controllers during evolution.

Usage:

```
java -jar SEED_NetworkEvaluator.jar -net <network> <agentName> [<fitness>]
```

Example:

```
java -jar SEED_NetworkEcaluator.jar -net a-series_oscillator.smb humanoid
```

The provided network controller is a simple oscillator which is controlling some of the motors simultaneously to demonstrate the network interface.

References

[1] ORCS Home Page at the Neurocybernetics Group University of Osnabrï¿½ck. http://www.cogsci.uni-osnabrueck.de/ neurokybernetik/tools/orcs, 2008.

[2] Jan Bender. *Impulsbasierte Dynamiksimulation von Mehrkörpersystemen in der virtuellen Realität*. PhD thesis, University Karlsruhe, 2007.

[3] Christian Rempis and Verena Thomas. ORCS ODCP Specification 1.0. URL `http://www.ikw.uos.de/~neurokybernetik/files/orcs/ORCS_ODCP_Specification_1.0.pdf`.

[4] Christian Rempis and Verena Thomas. Orcs standard Protocol OSP 1.1, 2008. URL `http://www.ikw.uos.de/~neurokybernetik/files/orcs/ORCS_OSP_Specification_1.1.pdf`.

[5] ROBOTIS. Robot Kit "Bioloid". URL `http://www.robotis.com/zbxe/bioloid_en`.

[6] Christian Thiele. Integrierte Entwicklungsumgebung zur Bewegungssteuerung humanoider Roboter. Student Research Project, 2007.

[7] Trolltech. Qt Reference Documentation (Open Source Edition), 2005.